Automatic Type Deduction Solutions

Copyright © James Raynard 2021

auto

- What is meant by "auto" in Modern C++?
  - Traditionally, we wrote a variable's type, followed by its name and its initial value
  - In Modern C++, we can write "auto" instead of the type, and the compiler will deduce the variable's type from the initial value
- Why is auto important?
  - Allows us to write simpler definitions of variables with complex types
  - Allows us to define variables whose type cannot be known by us
- Write a program which defines variables using "auto". Print out the values of the variables

Copyright © James Raynard 2021

auto with qualifiers

- What happens when the initializer is a const int?
- What happens when the initializer is a reference to int?
  - In both cases, the underlying type is deduced, namely int
- Write a function which returns a reference to const int
- Write a program which creates a variable using auto and initializes it with the return value from this function
- Demonstrate that the created variable is not const
- Modify the program so that the variable is a const reference
- Demonstrate that now the created variable is const

Copyright © James Raynard 2021

Using auto to simplify your code

- Use auto to write down simpler versions of the following definitions

```
pair<set<int>::iterator, bool> ret = ints.insert(3);
for (map<string, int>::iterator it = scores.begin(); it != scores.end(); ++it) ...
```

  - Replace the type by "auto" and the compiler will deduce the type automatically

```
auto ret = ints.insert(3);
for (auto it = scores.begin(); it != scores.end(); ++it) ...
```

- Suppose we want to prevent the loop from changing the map elements in the second example. Is this possible?
  - We can do this by making the iterator const

```
for (const auto it = scores.begin(); it != scores.end(); ++it) {
```

Copyright © James Raynard 2021

decltype

- Explain the purpose of decltype
  - decltype allows us to create a variable whose type is the same as some expression
  - The expression is used as an argument to decltype
  - (The expression is not used to initialize the variable)
- Imagine we have a function func() that returns a value
- We want to create a variable x which has the same type as this return value
- Write down a definition of x which uses decltype to deduce x's type

```
decltype(func()) x;
```

Copyright © James Raynard 2021

decltype

- Write a function which returns a reference to const int
- Write a program which creates a variable using decltype and the return type of this function
- What is the type of this variable?
  - Its type is reference to const
- Demonstrate whether the variable is const

Copyright © James Raynard 2021

decltype and references

- What happens if the argument to decltype is const?
  - The deduced type will be const
- What happens if the argument to decltype is a reference?
  - The deduced type will be a reference
- In the following code, what is the deduced type of j?

```
int i;
decltype(i) j;
```

  - j has type reference to int (not int)
- Write down a definition which produces the expected result

```
decltype((i)) j;                // j has type int
```

Copyright © James Raynard 2021

decltype(auto)

- Explain the purpose of decltype(auto)
  - This is similar to auto, but it retains const, reference, etc
- Write a function which returns a reference to const int
- Write a program which creates a variable using decltype(auto) and initializes it with the return value from this function
- Demonstrate that the created variable is const

Copyright © James Raynard 2021

Functions returning auto

- What does it mean for a function to return auto?
  - The compiler will deduce the return type of the function from the returned value
- Is this a desirable thing to do?
  - Not generally - the return type is part of the function's interface and hiding it removes information which is useful to programmers
  - Can cause problems if the deduced type is not what the caller expects
  - However, useful when the returned type cannot easily be determined by the programmer

Copyright © James Raynard 2021